BigInt:

基础数据类型，BigInt 只用来表示整数，没有位数的限制，任何位数的整数都可以精确表示。

```
 //定义一个bigint类型
 let bn = 100n;
     console.log(typeof bn);//bigint
//转换成bigint类型
     console.log(BigInt(200));//200
```

BigInt 可以使用负号（-），但是不能使用正号（+），因为会与 asm.js 冲突。

```
-42n // 正确
+42n // 报错
```

BigInt 函数

JavaScript 原生提供`BigInt`函数，可以用它生成 BigInt 类型的数值。转换规则基本与`Number()`一致，将其他类型的值转为 BigInt。

```
BigInt(123) // 123n
BigInt('123') // 123n
BigInt(false) // 0n
BigInt(true) // 1n
```

`BigInt()`函数必须有参数，而且参数必须可以正常转为数值，下面的用法都会报错。

```
new BigInt() // TypeError
BigInt(undefined) //TypeError
BigInt(null) // TypeError
BigInt('123n') // SyntaxError
BigInt('abc') // SyntaxError
```

参数如果是小数，也会报错。

```
BigInt(1.5) // RangeError
BigInt('1.5') // SyntaxError
```

函数默认值：

为函数设置默认值，可直接写在参数定义的后面，只有在未传递参数，或者参数为 undefined 时，才会使用默认参数，null 值被认为是有效的值传递

```
function strconcat(str1 = '', str2 = ''){//函数默认值可以直接写在参数后
 //ES5的写法
 /**
    str1 = str1 || '';
    str2 = str2 || '';
 */
          console.log(str1 + str2);
  }
     strconcat('abc', 'def');
     strconcat('abc');

// num为默认参数，如果不传，则默认为10
function fn(type, num=10){
    console.log(type, num);
}
  fn(1);  // 打印 1，10 （默认参数不传，则默认为10）
  fn(1,2); // 打印 1，2 （此值会覆盖默认参数10）
```

rest参数：

rest参数（“...变量名”），用于获取函数多余的参数（接收的是除去形参之外的所有的实参）

```
function sum(n, ...value) {
       console.log(arguments);
       //[100, 200, 300, 400] argument 获取所有的参数
       console.log(value);
       //[200, 300, 400] 获取函数多余的参数
   }
       sum(100, 200, 300, 400);

function sum(n, ...value) {
    console.log(value.length);
  }
    sum(100, 200, 300);//2
```

```
    sum(100, 200, 300, 400);//3
```

箭头函数：

基本语法：参数 => 函数体

a. 基本用法

```
let fn = v => v;
//等价于
let fn = function(num){
    return num;
}
fn(100);  // 输出100

/**
1、如果只有一个形参，()可以省略；
2、其他情况(没有参数)都不能省略；
**/
let f = n => { console.log(n * n); };
    console.log(typeof f);
    f(10);

/***
1、如果只有一个语句并且是返回值，{}和return都可以省略；
*/
let pf = i => i * i;
     console.log(pf(20));

var arr = [2, 3, 4];
var newArr = arr.map(i => i * i);
    console.log(newArr);
```

b. 带参数的写法

```
let fn2 = (num1,num2) => {
let result = num1 + num2;
    return result;
}
fn2(3,2);  // 输出5
```

c. 箭头函数中的this指向问题

箭头函数体中的 this 对象，是定义函数时的对象，而不是使用函数时的对象。在函数定义的时候就已经决定了

```
function fn3(){
    setTimeout(()=>{
// 定义时，this 绑定的是 fn3 中的 this 对象
    console.log(this.a);
  },0)
}
var a = 10;
// fn3 的 this 对象为 {a: 10}，因为它指向全局: window.a
   fn3.call({a: 18});  // 改变this指向，此时 a = 18

//-------------------------1
function f() {
     console.log(this);//window window在调用
   }
   f();

//-------------------------2
let obj = {
    a: 1,
    f: function () {
         console.log(this.a);
       }
 };
   obj.f();//1
let f1 = obj.f;
    f1();//没有对象在调用，默认的是window  window.a --->  undefined

//-------------------------3
var name = 'lili'; //var声明的变量会追加到window对象身上
var obj = {
    name: 'liming',
    prop: {
         name: 'ivan',
         getname: function () {
             return this.name;
```

```
        }
    }
};
    console.log(obj.prop.getname()); //ivan obj.prop在调用
 var test = obj.prop.getname;
    console.log(test());//lili

//-------------------------4
let obj3 = {
    a: 3,
    f: function () {
        console.log(this.a);
        let a = 5;
        return () => console.log(this.a);
     }
};
let f3 = obj3.f();//3
    f3();//3  箭头函数的this在定义的时候已经确定了，跟调用没有关系
let f4 = obj3.f;
    f4(); //undefined
    f4()();//undefined undefined

//-------------------------5
var a = 500;
let obj4 = {
    a: 4,
    f: function () {
        console.log(this.a);
        let a = 6;
            return {
               a: 7,
               f: () => console.log(this.a)
            };
        }
};
let obj41 = obj4.f();//4
    obj41.f();
let obj4f = obj4.f;
    obj4f().f();
```

箭头函数有几个使用注意点：

（1）函数体内的this对象，就是定义时所在的对象，而不是使用时所在的对象。

```
// ES6代码
function foo() {
   setTimeout(() => {
    console.log('name:', this.name);
  }, 100);
}

// 转为 ES5代码
function foo() {
   var _this = this;

   setTimeout(function () {
    console.log('name:', _this.name);
  }, 100);
}
//箭头函数里面没有自己的this，而是引用外层的this
```

（2）不可以当作构造函数，也就是说，不可以使用new命令，否则会抛出一个错误。

（3）不可以使用arguments对象，该对象在函数体内不存在。如果要用，可以用 rest 参数代替。

```
let fn = () => console.log(arguments);
    fn(1, 2, 3);
//报错: Uncaught ReferenceError: arguments is not defined

function foo() {
  setTimeout(() => {
    console.log('args:', arguments);
  }, 100);
}

foo(2, 4, 6, 8)
// args: [2, 4, 6, 8]  箭头函数内部的变量arguments，其实是函数foo的arguments变量
```

（4）不可以使用yield命令，因此箭头函数不能用作 Generator 函数。

函数的扩展 - ECMAScript 6入门 (ruanyifeng.com)